Exploiting Agent Mobility for Large-Scale Network Monitoring

Antonio Liotta and George Pavlou, University of Surrey
Graham Knight, University College London

Abstract

As networks become pervasive, the importance of efficient information gathering for purposes such as monitoring, fault diagnosis, and performance evaluation increases. Distributed monitoring systems based on either management protocols such as SNMP or distributed object technologies such as CORBA can cope with scalability problems only to a limited extent. They are not well suited to systems that are both very large and highly dynamic because the monitoring logic, although possibly distributed, is statically predefined at design time. This article presents an active distributed monitoring system based on mobile agents. Agents act as area monitors not bound to any particular network node that can "sense" the network, estimate better locations, and migrate in order to pursue location optimality. Simulations demonstrate the capability of this approach to cope with large-scale systems and changing network conditions.

As the size of networked systems grows, it becomes increasingly difficult to extract the right information from them. Networked systems and even networks themselves need constant monitoring and probing for the purposes of management, particularly for fault diagnosis and performance evaluation. For instance, network monitoring entails the collection of traffic information used for a variety of performance management activities such as capacity planning, bottleneck and congestion identification, quality of service monitoring for services based on service level agreements, and so on. A key aspect is that the collection of traffic information should be supported in a timely manner so that reaction to performance problems is possible; in addition, monitoring traffic should ideally have a minimal impact on the managed network.

In order to determine whether existing monitoring systems can satisfy the requirements imposed by future "networked" systems, we need to first identify these requirements. An important difference between current and future networking is that in the former case *topology* is considered relatively static, while this becomes an unsafe assumption for future networks. New models of dynamic networks are contrasting with the situation in which the network topology was mainly modified as a result of careful planning. Examples in which logical topology layering can be dynamically constructed in real time are the following:

- Dynamically reconfigurable networks
- Active networks
- Dynamic virtual private networks (VPNs)
- Mobile and survivable networks
- Reconfigurable cellular networks

The work reported in this article has formed part of the Software Based System work area of the Core 2 Research Program of the Virtual Centre of Excellence in Mobile & Personal Communications, Mobile VCE, http://www.mobilevce.co.uk

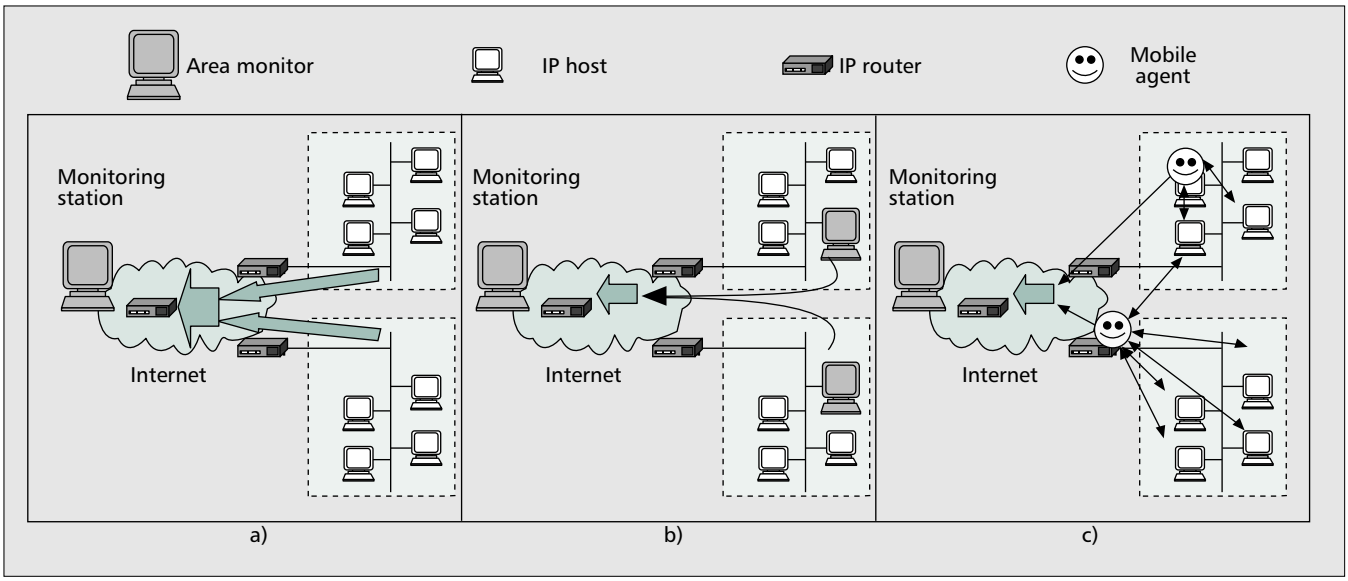
Another important feature of future networks is *scale*. The tremendous success of the Internet has made possible and even encouraged the realization of systems characterized by very large scale, and high levels of distribution and dynamics. "Network-centric" approaches such as Sun's Jini architecture envisage large numbers of comparatively simple devices (cellular phones, televisions, thermostats) all accessible through the network. Management systems in the future will need to keep track of these devices and determine which are present, which are functioning correctly, and so on.

The third key factor is the appearance of increasingly complex services and applications that rely on heterogeneous integrated fixed and mobile networks. The variety of such applications and services along with the possibility of accessing them from virtually any location makes it extremely difficult to anticipate the type and distribution of *network traffic*. This in turn implies that network traffic and potential congestion may appear anywhere.

Because of their scale and dynamics, future networked systems (as well as applications and services) will be very difficult to manage and control unless efficient means of monitoring them become available. Monitoring systems will need to efficiently gather network and system state on a large scale in order to feed back information to management and control applications.

In this article we argue that the conventional approach to network monitoring, based on either *management protocols* or *distributed object* technologies, cannot fully satisfy the requirements of future networked systems explained above. Intuitively, in order to monitor large-scale dynamic systems we need a distributed monitoring model that adapts to the monitored system. After discussing the limitations of monitoring models in the Simple Network Management Protocol (SNMP), telecommunications management networks (TMN), and distributed object approaches such as the Common Object Request Broker Architecture (CORBA) and Java Remote

 0890-8044/02/$17.00



■ Figure 1. *Network monitoring models: a) static centralized monitoring; b) static decentralized (hierarchical or based on distributed objects) monitoring; c) dynamic (i.e., programmable or active) decentralized monitoring with MAs.*

Method Invocation (Java-RMI), we discuss the potential of employing mobile agents (MAs) in future network monitoring systems. An MA is essentially an autonomous object containing the logic to perform a given task and possibly migrate under its own control from node to node in a network [1].

The idea of employing MAs for applications such as distributed information retrieval, performance monitoring, and remote data filtering or aggregation has been extensively reported in the literature; an interested reader may, for instance, refer to the state-of-the-art review presented in [2]. However, most commonly, in the context of management, MAs are not exploited to the full potential of their capabilities. The majority of examples presented use MAs simply as a mechanism to realize dynamic programmability of remote elements according to the management by delegation (MbD) concept [3].

MbD may be realized with agents bound to *single-hop* mobility; the agents move from the managing node to remote managed nodes, and then stay put. What is not commonly exploited in management is the agent *multiple-hop* capability where agents may move several times as they adapt to changing circumstances. While single-hop mobility can provide increased flexibility and scalability in the context of relatively static networked systems, it is the multiple-hop capability offered by MAs that needs to be exploited to meet the requirements of future networked systems (i.e., large scale and dynamics). In this article we support this argument through an example MA-based system that exploits full code mobility to realize a scalable and adaptable distributed monitoring model.

Given our proposal to use MAs as area monitoring stations, a distributed algorithm is required to compute the agent locations, initially offline and then at runtime. During the execution of the monitoring task, agents will need to sense their environment and take actions in order to adapt to changing conditions and, by doing so, maintain location optimality. Optimality in this case concerns the minimization of network traffic incurred by the agent-based monitoring system and of latency in collecting the necessary information.

Our algorithm relies on agents learning about the network topology through node routing table information accessed through standard management interfaces. The monitoring system is initially deployed through a *clone and send* process starting at the centralized network-wide station. The same algorithm is also used by the agents to adapt to network changes through migration. Key features of this algorithm are its distributed nature (i.e., each agent carries and runs the algorithm) and low computational complexity. We use simulations to assess the scalability of our approach, and the ability to adapt to network congestion and faults.

Network Monitoring Techniques

Static Centralized Monitoring

In this case there is a single monitoring station with which all monitored systems communicate directly. The monitoring station is in charge of collecting, aggregating, and processing raw network data (Fig. 1a).

The static centralized model is widely used to manage relatively static small-scale networks (i.e., private networks) using SNMP. The model has been criticized for its limited responsiveness and accuracy, and lack of scalability. The concentration of management intelligence in a single point results in processing and communication bottlenecks, limiting the number of elements that can be monitored and the rate at which information can be gathered. Furthermore, SNMP favors a polling approach which limits the ability to track problems in a timely manner while requiring management traffic even if no significant change has occurred.

To overcome the shortcomings of polling, the alternative technique of event reporting may be used. With event reporting the monitored systems take the initiative to inform the manager according to predetermined rules set up by the manager. Event reports are conceptually generated within the monitored systems, either periodically or when a particular condition is reached. *Periodic reporting* provides the manager with status information, typically in a summarized fashion, and it is more efficient than requesting the same information via polling. On the other hand, *alarm reporting* is useful for detecting problems as soon as they occur. The problem with alarm reporting is that the types of alarms need to be thought out in advance, standardized, and supported by vendors. Event reporting requires an increased level of intelligence in the monitored systems.

Typical systems employ both polling and event reporting, although in practice telecommunications management systems rely more on event reporting, and private network SNMP-based management systems use mostly polling. For simplicity in the remainder of this article we will assume that monitoring tasks are implemented with the polling mechanism.

Static Decentralized Monitoring

One way to pursue increased performance and scalability is to adopt a *hierarchical* management architecture, which uses multiple systems with one system acting as a main monitoring station and the others working as *area monitors* (Fig. 1b). Hierarchical monitoring is used in the TMN. In the context of SNMP, simple monitoring and statistical probes can be introduced using RMON, which is equivalent to an area monitor that collects monitoring information about a number of elements within a subnetwork. More recently, other forms of decentralization based on distributed object technologies such as CORBA and Java-RMI have become popular in management. An extensive review of management paradigms and technologies can be found in [4].

The common denominator of the above approaches is the adoption of simple, predefined functionality that can result only in a limited level of decentralization of management intelligence. Monitoring functionality that can actually be decentralized is restrained to operations such as low-level filtering of monitoring data, generation of alarms on the basis of simple conditions, and collection of rudimentary statistical information. In addition, these decentralized area monitors operate in predefined network locations, which means that they cannot easily adapt to network changes. Therefore, conventional static decentralized schemes, despite coping with the scalability problem to a certain extent, inherit the other problems of centralized management and cannot easily cope with frequently changing dynamic environments.

Programmable Decentralized Monitoring

What is needed is a mechanism to dynamically deploy new management logic when and where needed without having to predefine that logic. With distributed object technologies, the management logic can only be modified through software reinstallation.

The first proposal to support remote programmability was introduced by Yemini *et al.* with their MbD framework (see [3], although the original paper was published in 1991). MbD also represents one of the first concrete attempts to make use of mobile code in network management, signaling a paradigm shift from static to dynamic management. The basic underlying principle is that new management functions can be dynamically introduced to a managed node as required. The manager uses the MbD protocol to push new code down to the managed node; management routines are executed locally rather than centrally at the management station. Therefore, this is a mechanism to decentralize management processing and reprogram managed node capability.

When MbD was first proposed Java did not exist, and the idea of pushing executable code on the fly presented several hurdles. Managed nodes were relatively simple in terms of processing power, and there was no uniformity in processing environments. It is the increase in processing power and the advent of Java that have made the MbD paradigm a viable solution. Java's object serialization makes it easy to migrate code, while Java-RMI provides for simple communication between distributed objects. The single-hop mobility mechanism envisioned by MbD, despite being extremely useful as a mechanism for flexible and dynamic remote programmability, is still a relatively static mechanism since it is only used to deploy management logic at startup time. The decisions of when and where to deploy management logic are still made by a centralized management station based on a static network view. Because the MA is conceived as a dynamically deployable piece of code (single-hop mobility) rather than free to roam the network (multiple-hop migration), full code mobility is not exploited to provide runtime adaptation. Therefore, the single-hop mobility mechanism does not fully satisfy the requirement of large-scale highly dynamic networked systems.

Active Distributed Monitoring

The possible advantages of using agent mobility for network management have been discussed extensively in the agent community and are reviewed in [2]. Some of these advantages are reduced network traffic, and increased responsiveness and robustness.

The question addressed in this article is how to exploit agent multiple-hop mobility to build a distributed monitoring system that reconfigures itself as the status of the monitored system changes (Fig. 1c). Reconfigurability is an essential requirement if the status of the monitored system is dynamic and transient. We have seen that with distributed objects and single-hop mobility we can only realize a relatively static monitoring system that may or may not be optimized on the basis of the initial status of the monitored system. As the latter evolves, the distributed monitoring logic may have to be relocated in order to maintain optimality; when MAs are used as adaptive area monitors, their optimal locations depend on the status of the network, which may vary considerably in highly dynamic environments.

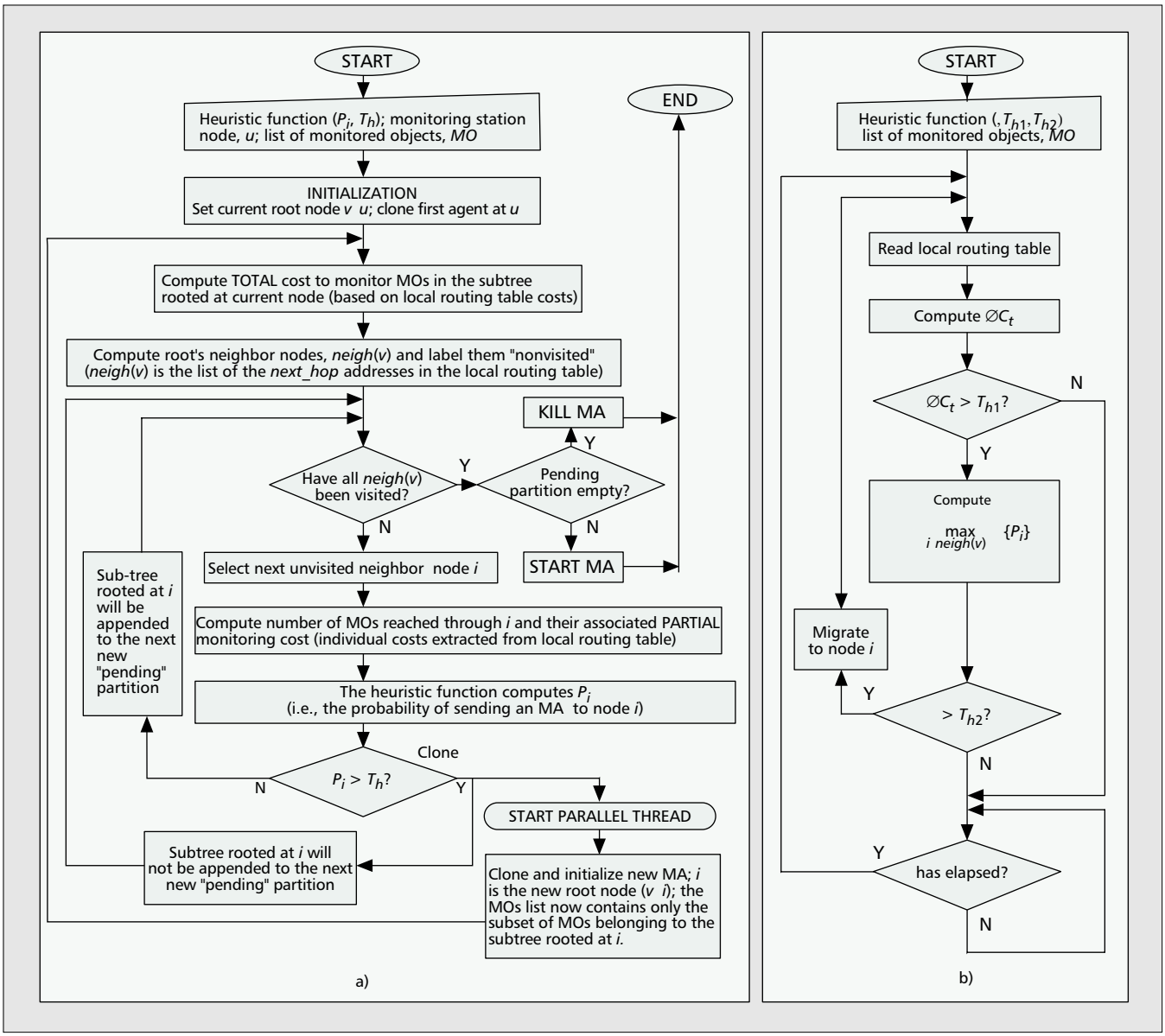
In the rest of the article we present an example of an agent system that realizes *active distributed monitoring*. The system is *decentralized* because the monitored system is partitioned and separate agents are dynamically assigned to disjoint partitions. Network partitioning is computed in a *distributed* fashion by the agent system. Finally, because agents are capable of sensing the network status and migrating at runtime to maintain location optimality, the system is *active* or adaptive. Such a system exploits not only multiple-hop mobility but also *agent autonomy* (each agent contains the logic to independently decide when and where to migrate) and *agent cloning*, the ability of an agent to create and dispatch copies of itself.

Effective Location of Area Monitors

The agent location problem consists of two phases. Initially, we need to determine the appropriate number of agents for a given monitoring problem and compute the location of each of those agents. Subsequently, upon agent deployment the agent system needs to be able to self-regulate in order to adapt to changing conditions. This is achieved by triggering agent migration in a controlled fashion to avoid instability due to continuous agent migration.

The problem of computing the optimal number and location of area monitors is analogous to the optimal placement of p servers in a large network, which has been studied since the early '70s. This belongs to the class of p-center and p-median problems, both NP-complete when striving for optimality [5]. Approximate polynomial algorithms have been proposed, but none of them suits the requirement of our agent system. Proposed algorithms are centralized, requiring the network distance matrix at the main monitoring station. While this is less of a problem in offline calculations for medium to long-term optimal locations, it becomes an important problem for active distributed solutions in which optimal locations need to be (re)calculated by the agents themselves. In this case, the monitoring station should retain an up-to-date version of the whole network topology, which obviously is an unrealistic requirement for large-scale dynamic networked systems.

In the system proposed herein, the location of area monitors is neither fixed nor predetermined at design time. Area monitors are realized with MAs, simple autonomous soft-



■ Figure 2. *Agent configuration algorithms: a) agent deployment; b) agent self-relocation.*

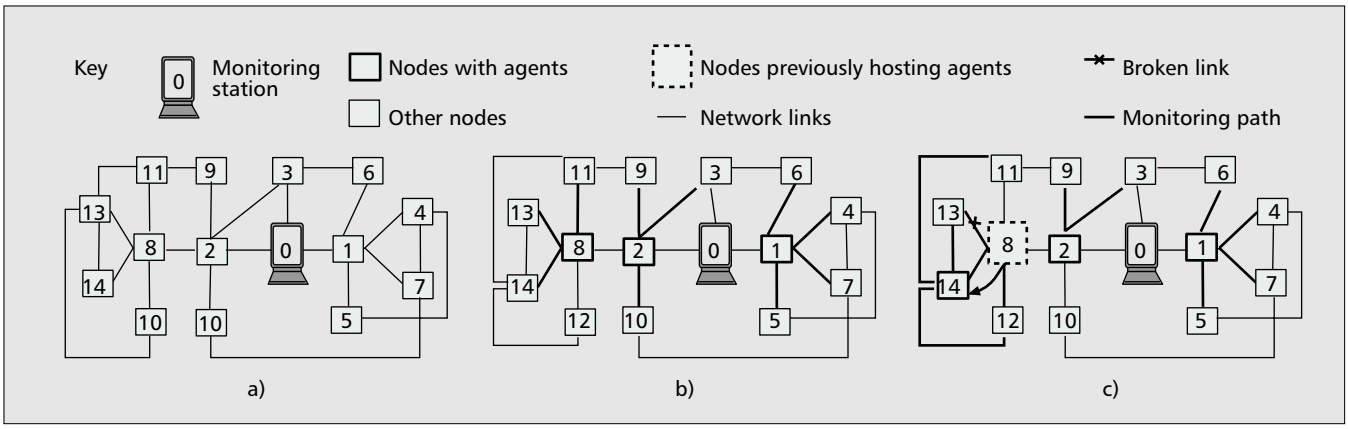
ware entities that, having access to network routing information, can adapt and roam through the network. The distributed monitoring system is deployed by progressively partitioning the network and populating each partition with monitoring agents.

We assume the existence of an agent system supporting *mobility* and *cloning*. Agents are assumed to have access to routing information obtainable from network routers through standard network management interfaces. For simplicity, we also assume that MA hosts (i.e., locations in which MAs are able to run) are evenly distributed within the network. This, in other terms, means that for each router there is always an MA host that is located relatively close to it and, for each LAN, the number of MA hosts is proportional to the number of monitored objects (MOs) that need to be monitored in that LAN. Under these assumptions, the MA distribution tree (i.e., the set of routes used for MA deployment) does not differ significantly from the routing tree rooted at the monitoring station. Without loss of generality, we envisage a scenario in which routers can act as MA hosts during MA deployment. In such case, the MA distribution tree would actually coincide with the routing tree.

Agent Deployment

The conventional approach to partitioning the monitored system on the basis of its topological and dynamic features is clearly not viable in the case of very-large-scale dynamic systems. The task of collecting a real-time snapshot of the topology and traffic profile of a large-scale rapidly changing networked system is indeed ambitious. The basic underlying idea used to solve the agent location problem is to exploit information that is readily available in the system rather than trying to derive a new network distance matrix. This is why our monitoring system relies solely on routing information, which is maintained by routing protocols. The precise nature and quality of this information will depend on the routing protocol in use. Sophisticated routing protocols such as Open Shortest Path First (OSPF) can maintain information using multiple distance metrics (delay, throughput, etc.). Simple routing protocols such as RIP, however, use only a hop-count metric. Our objective is to design a deployment algorithm that will optimize agent location with respect to whatever metric information is available. However, it is clear that performance of our system, in absolute terms, will be affected by the quality of this information.

A flow chart of the proposed agent deployment algorithm



■ Figure 3. *a) Sample network topology; b) final agent location; c) example adaptation through agent migration, following a link failure.*

is depicted in Fig. 2a. The algorithm is illustrated through a simple example for the network depicted in Fig. 3a. For the sake of simplicity, we assume that the list of MOs consists of all the network nodes. The basic phases of the agent location process are depicted in Fig. 4. The algorithm deploys the area monitors (or MAs) during the network partitioning process through a "clone and send" process starting at the main monitoring station. The *number* and *location* of MAs is computed by subsequently comparing the monitoring task parameters with routing information extracted from network routers.

The monitoring task, including the list of MOs as well as the operations to be performed on them, is delegated to a first MA (Fig. 4a). This MA starts executing the algorithm of Fig. 2a, which is distributed in nature; that is, each subsequently cloned MA will execute the same algorithm (only the input configuration parameters will differ). Starting from the monitoring station, the initial MA builds an estimate of the *total* monitoring cost on the basis of the individual cost or *distance field* of the local routing table (i.e., for each element of the MO list, its routing cost will be added to the total estimated monitoring cost). Having estimated the total cost for performing the task from its current location, the MA will then consider alternative configurations using *partial* costs attached to the neighbor nodes. Not all neighbor nodes need to be discovered: only those nodes that are next hops for the MOs targeted by the MA are candidate MA hosts. They will be discovered, again, through the local routing table; that is, they are identified by next-hop addresses of those entries whose *destination field* contains one of the targeted MOs. The partial cost attributed to a neighbor node is found by adding the individual costs (extracted from the table) of those MOs that are targeted by the MA and reached through the node. At this point, a simple heuristic function is used to decide the *number* of new agents to be cloned and their initial *location* (among the neighbor nodes). This function estimates the need to send or clone an MA to a neighbor (i) on the bases of the total number of MOs targeted by the agent ($|MO|$), the number of MOs reached through the node ($|MO_i|$), the total cost (C_t), and the partial cost (C_i). The cloning threshold value (T_h) impacts the final number of MAs. In our system, T_h has been tuned in order to achieve a percentage number of MAs on the order of 5–10 percent of the total number of network nodes. The probability of an agent being sent to a given neighbor (i) has been computed using the following formula:

$$P_i = \frac{|MO_i|}{|MO|} * \frac{|C_i|}{|C_t|}.$$

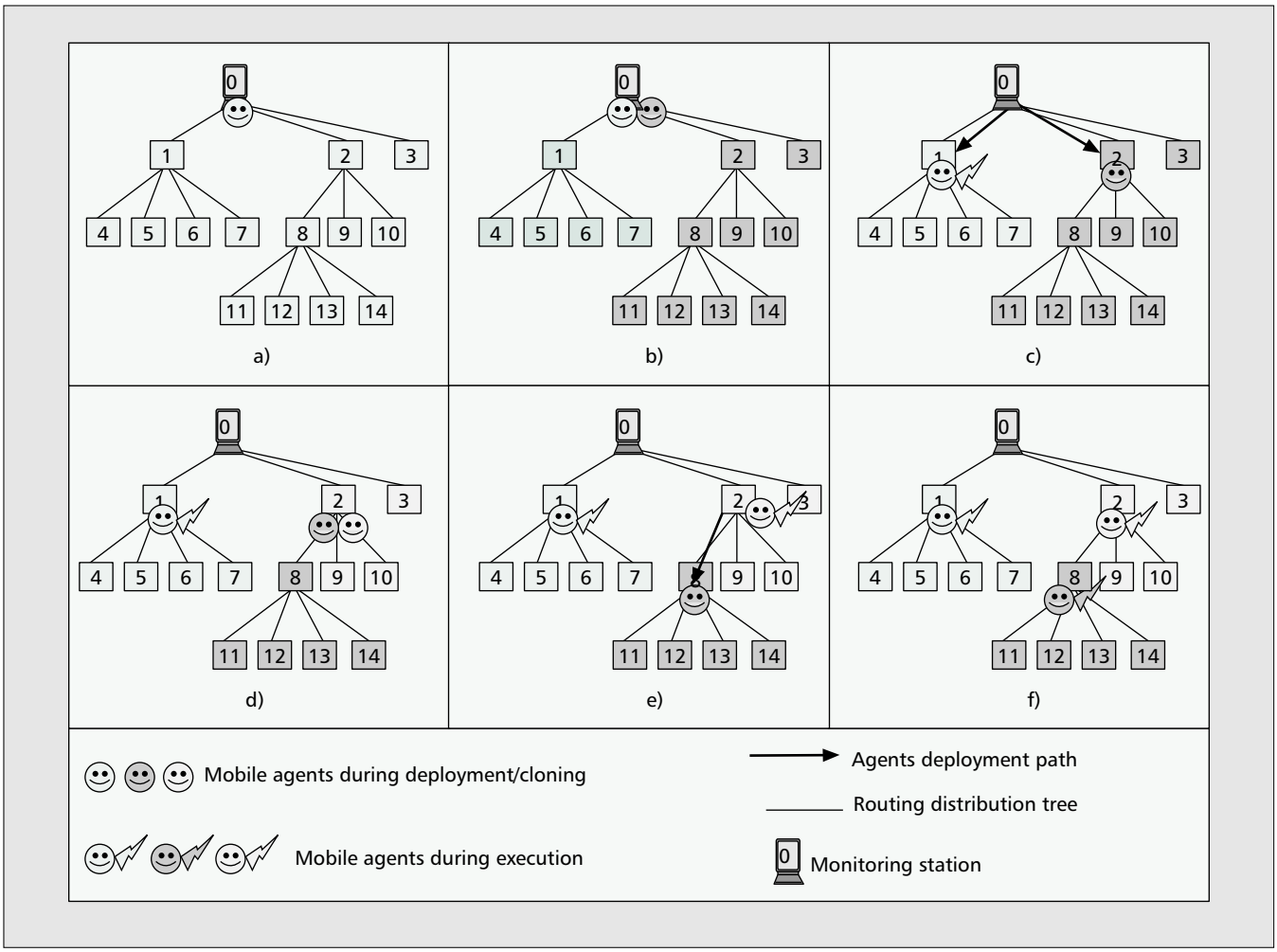
In our example, the root agent (sitting at location 0) estimates that probabilities P_1 and P_2 associated with neighbors 1 and 2, respectively, are sufficiently high to justify sending an agent to node 1 and another to node 2 (Fig. 4b). P_3 is relatively low (only one node is monitored through node 3). Therefore, the network is partitioned in two portions; two agents are needed: one agent is available already, one extra agent is cloned; and the list of MOs of each agent is reset in order to reflect the new network partitioning. Finally, the agents are deployed to their new location (Fig. 4c) and each of the two agents independently continues the clone and send process. The agent sitting at node 1 is now ready to start monitoring its subpartition since P_4–P_7 are relatively small. In contrast, the agent sitting at location 2 decides to produce a further subpartition and clones a new agent; P_8 is above threshold (Fig. 4d). The decomposition/cloning/migration process continues in a similar way (Fig. 4e), leading to the final configuration of Fig. 4f. Figure 3b depicts the agent location within the network (we can observe that the monitoring path, i.e., the agent-to-monitored nodes communication path, does not necessarily coincide with the agent deployment path).

It should be noted that the use of cloning results in minimal agent deployment traffic around the monitoring station. In fact, only two agents leave the station, although the resulting number of agents is three. The cloning algorithm is executed in a distributed fashion (on nodes 1, 2, and 8). Finally, the processing is performed in parallel among nodes at the same levels (1 and 2). This algorithm is computed dynamically in the sense that the final agent location depends critically on the network status detected at deployment. Clearly, the effectiveness of the heuristic function in placing MAs is affected by the quality of the routing tables whose values are sensitive to the routing protocols operating in the network (independent of the agent system).

Keeping Pace with System Dynamics

Upon initial agent deployment, the system still needs to be able to adapt to changing network conditions. For instance, node failures, link congestion, and mobile computing result in rapidly changing logical topologies. Conventionally, dynamic routing protocols react to such changes by appropriately manipulating the relevant routing tables in order to bypass problematic portions of the network. The impact of such an approach on a static monitoring system will be that its monitoring packets will be rerouted through generally longer paths, causing both traffic and response time to deteriorate.

With the proposed adaptive monitoring system, upon deployment each agent keeps sensing the network by periodically accessing the local routing tables and reconsidering its location. The flow chart of Fig. 2b depicts the agent self-relocation process. The underlying principles are very similar to those used at deployment time. However, to avoid uncontrolled agent proliferation, agents adapt through relocation but do not adopt



■ Figure 4. *Example agent deployment process (agent deployment path).*

cloning. The parameters of the heuristic function are slightly different. One parameter is the table *read up* period, Π (i.e. the interval between two successive accesses to the local routing table). At each interval, the agent will recompute the total cost associated with its MO list and the gradient cost, ΔC_t, with respect to the previews interval. A second parameter is the threshold, T_{h1}, applied to ΔC_t, which initiates the migration decision process. Partial costs are then computed for the neighbors, similar to the deployment phase, and the node with maximum deployment probability is identified. A second threshold, T_{h2}, is applied to that node in order to make a final decision on whether or not to trigger migration.

A simple example illustrating agent self-regulation in response to a link failure is depicted in Fig. 3c. In this case, following a loss of connectivity between nodes 8 and 13, a new (longer) monitoring path is established between nodes 8 and 13. As a result, the central node for the system partition comprising nodes {8, 11, 12, 13, and 14} becomes node 14. Hence, the agent originally located in node 8 will relocate to node 14, bringing the system back to optimality.

Evaluation Methodology

The proposed agent-based monitoring system has been evaluated from different viewpoints: *scalability*, *optimality*, and *adaptability*. The complete analysis was detailed in the Ph.D. thesis of one of the authors [6]; in this article we report the most significant findings.

Scalability essentially concerns two aspects. The first regards the evaluation of the *computational complexity* of the agent deployment process and estimation of its associated timescale. This aspect has been assessed through mathematical analysis. The other aspect concerns the actual performance of the system at *steady state* (i.e., in the initial agent deployment phase). Performance in terms of incurred monitoring *traffic* and *response time* has been assessed against various scalability factors: *network size* (number of nodes and network diameter), monitoring *polling rate*, and *number of MAs to number of MOs ratio*. Steady state scalability has been assessed through simulation. The location algorithm has been run for a set of realistic network topologies composed of routers, links, and hosts. These have been generated using the GT-ITM topology generator following a well established methodology by Calvert and Zegura [7, 8]. In particular, transit-stub topologies resembling Internet topology and with a range of 16–100 nodes have been generated in order to assess the sensitivity of the location algorithm to network size.

In order to simulate IP network and protocol behavior we have adopted the NS-2 simulator from the University of California at Berkeley/Lawrence Berkeley National Laboratory [9] and extended it with MA capabilities. Agent migration and cloning have been implemented along with the actual agent location algorithm, which is incorporated in each agent. This algorithm has been configured to minimize the total incurred monitoring traffic.

After quantifying the scalability of our system in terms of traffic and response time, we have conducted comparative simulations between our agent deployment algorithm and the

Lagrangian location algorithm [5]. The latter is not a viable solution (it does not scale) since it relies on the network distance matrix being made available at the centralized station. It is used merely because it leads to provably near-optimal agent configurations. Those configurations have been used as a reference point for the assessment of the distance of our system from optimality. We also generated the agent location randomly to emulate an unoptimized agent distribution.

Finally, we studied the reconfigurability of our agent system by simulating various conditions in which link failures led to increased traffic and response time. We deployed the agent system before the failures, then generated link failures at random locations, assessed the costs associated with agent migration, and finally measured traffic and response time after reconfiguration.

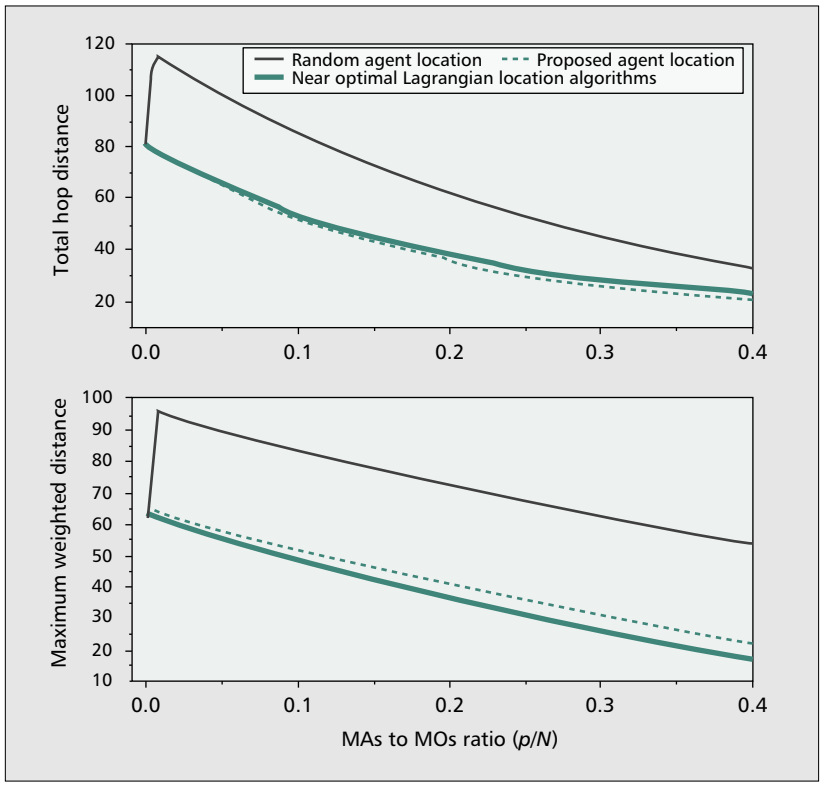


Figure 5. *Distance from near optimality.*

System Assessment

Agent Deployment Timescale

Rather than delving into the details of the mathematical analysis reported in [6] we shall try, instead, to draw conclusions on the agent deployment timescales following a more intuitive approach. If we look again at the example deployment process depicted in Fig. 4 we can see that, because agent deployment happens together with network partitioning, the system does not need to collect and process any network status information. Agents base their decision on information solely contained in the local routing tables. On the other hand, agent cloning and migration introduce significant overhead on the order of seconds [10]. Thus, if we assume that the processing power available at each agent site is sufficient to process the local routing table in a matter of milliseconds, we can conclude that the bottleneck of the system will be localized at the cloning/migration process. Finally, agents running at the same level of the distribution tree (e.g., agents sitting at nodes 1 and 2 in Fig. 4c) execute independent of each other and in separate physical locations (i.e., in parallel). Therefore, under the above assumptions agent deployment time increases linearly with the number of levels of the distribution tree; that is, the timescale of the deployment process is on the order of R s, whereby R is the network radius. For the example of Fig. 4 the deployment time is on the order of 2 s: 1 s to deploy the first two agents (Fig. 4c) plus 1 s to deploy the third agent from node 2 to node 8 (Fig. 4e).

Steady-State Performance

The performance comparison between the agent system and its centralized counterpart (i.e., a centralized static monitoring system) was aimed at quantifying the performance delta rather than proving the superiority of the former solution (expectedly, agents lead to reduced traffic and response time due to their intrinsic distributed nature). Because the simulations were conducted with lightly loaded networks, it was possible to grasp the effect on performance of various scalability factors. Both traffic and response time increased linearly with polling rate, number of MOs, and network diameter up to a certain *breaking point* (i.e., the point at which the network could not sustain the monitoring traffic). Under the simulated conditions, the agent solution led to an average 50 percent reduction in overall traffic and 30 percent reduction in response time. Due to their distributed nature, agents proved to be extremely efficient at dragging traffic away from the centralized network-wide monitoring station; the traffic incurred around the monitoring station by the agent system was negligible.

Another important scalability factor is the maximum polling rate sustainable by the network (this is related to the *accuracy* and *timeliness* achievable by a polling-based monitoring system). The agent system could sustain polling rates on the order of 200 percent larger than its centralized counterpart.

Finally, we assessed the sensitivity of the system to the number of MAs (keeping all other parameters unchanged). This time the performance curves exhibited nonlinear behavior. Both traffic and response time decreased significantly when the number of MAs to number of MOs ratio was smaller than 0.15. However, for larger fractions the performance improvement was negligible, showing that increasing the number of MAs is advantageous only up to a certain point. Beyond that point agent deployment overheads predominate.

Distance from Optimality

The distance from optimality of the proposed location algorithm can be evaluated by observing the plots of Fig. 5, where p is the number of MAs and N is the number of network nodes. The total hop distance is directly related to the total steady-state monitoring traffic. It can be observed that the proposed location algorithm leads to traffic values that are always smaller than those that would be achieved with the Lagrangian algorithm, which is provably near optimal [5]. Hence, our agent-based algorithm is near optimal too. In particular, a percentage improvement in the range of 0–3 percent was measured.

The fact that the total hop distance achieved by placing the agents in a random fashion is very far from our near optimal solution (38–48 percent difference for $p/N < 0.1$) provides another good justification for the adoption of the agent-based approach. The percentage reduction in traffic with respect to centralized polling ($p/N = 0$) is also significant. For instance, for $p/N = 0.1$ the reduction in traffic will be greater than 30 percent and will increase monotonically with p/N.

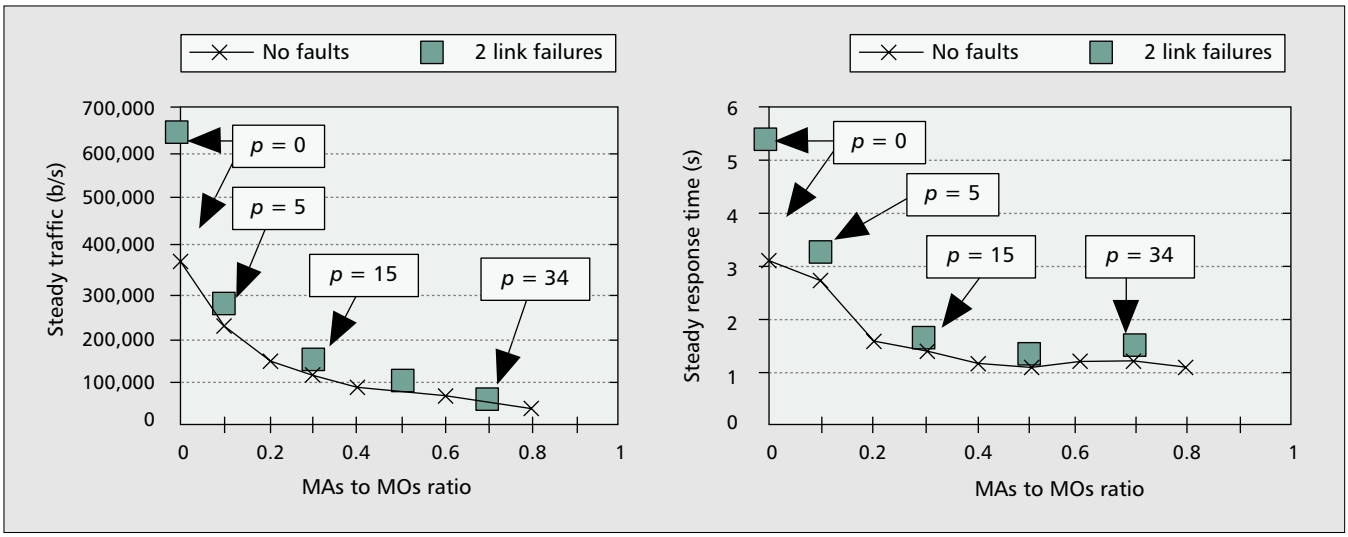


Figure 6. *Self-adaptation through agent migration.*

Finally, the fact that the three curves tend to converge for large values of p/N is not unexpected since when $p/N = 1$ the number of agents equals the number of nodes. Hence, each of the three location algorithms will equally succeed in placing the agent evenly. The plot that reports the maximum weighted distance (directly related to response time) for the three location algorithms is qualitatively analogous to the previous one. However, in this case the agent location curve, although very close to the near optimal one, does not exhibit any inferior value. In particular, the distance from near optimality is 0–5 percent for $p/N < 0.1$. This result was expected since the simulated agent location algorithm was configured to minimize traffic, not response time.

Agent Migration

Adaptation through migration can be demonstrated through a simple link failure scenario. Steady-state performance is measured before the failure; then sufficient time is allowed for the system to adapt in terms of routing table manipulation (by the relevant protocol layers) and subsequent agent migration. Finally, performance is measured again as soon as the new steady state is reached. Figure 6 depicts those two performance states in terms of traffic and response time for different system configurations; that is, for a range of agents comprised between 0 (centralized monitoring system) to a maximum of 34. With the centralized polling solution ($p = 0$), most request and response packets are rerouted through longer paths after failures (we simulated various scenarios with two failures on average). Consequently, both traffic and response time increase significantly; they almost doubled in our scenario. On the contrary, with the agent system the performance degradation at steady state is on the order of 5–10 percent only.

Concluding Remarks

In this article we present our progress toward the design of a self-regulating distributed monitoring system based on MAs. First, we address the common question, "What is the real benefit of using MAs for monitoring?" Having designed and extensively simulated an adaptable monitoring system strictly based on key MA features — *multiple-hop code mobility, cloning*, *reactiveness*, and *autonomy* — we have demonstrated, through simulation, the *viability* as well as the benefits of such an approach. While scalability can be pursued with distributed object technologies, it is with MAs that we can address the combined requirements of scalability and adaptability.

An unexpected lesson came from the study of the *optimality* of the agent system. Because the heuristic function was designed with *simplicity* and *computational efficiency* in mind, the authors were not expecting to find out that agent deployment would be near optimal with respect to traffic. It would be interesting to investigate whether with simple modifications to the heuristic function it will be possible to achieve near optimality with respect to response time. Another important outcome of the simulations regards *adaptability*. Again the authors did not anticipate the degree of adaptability achieved through agent self-relocation. An interesting investigation avenue may be to consider increasing the intelligence of MAs and their heuristics to stretch adaptability even further. One possibility is to investigate agent *proactiveness* as a means to anticipate problems rather than just react to them.

A significant open problem is that of *stability*. Agent relocation should be governed by principles analogous to those of classic control theory in order to avoid instability, oscillation, or slow response to problematic conditions. If agent migration is supported by general-purpose MA platforms, its relatively high migration overheads also give an indication of the timescales over which adaptation might be effective. The agent system will be able to react to changes within timescales larger than 1 s since typical agent migration times are on the order of 1 s.

Clearly the simulation-based study presented in this article is just one step toward the practical application of MAs to network and system management. Its main aim was to provide quantitative results that may motivate and encourage further work in this area. In addition to being pursued from a theoretical viewpoint, the proposed agent work may be applied to a number of domains. The system has been designed in order to meet the monitoring and control requirements of large-scale highly dynamic networked systems. The key benefit of scalability makes it suited to real-time monitoring. By populating the system with a suitable number of agents it is, in fact, possible to provide upper bounds on response time (the more agents deployed, the smaller their "catching" area). The authors are currently investigating active monitoring in the context of next-generation service management. MOs include not only heterogeneous mobile terminals and their local resources, but also advanced services. The number of such MOs is potentially very large, and their location is difficult to anticipate: terminals are free to roam and services may be implemented with MAs. In contexts such as this, the application of active monitoring is expected to prove particularly beneficial. However, it is through the experience gained with future practical implementation that further benefits and shortcomings of active monitoring will be unveiled.

Acknowledgments

The work reported in this article has formed part of the WA1 area of the Core 2 Research Programme of the Virtual Centre of Excellence in Mobile & Personal Communications, Mobile VCE, http://www.mobilevce.com, whose funding support, including that of EPSRC, is gratefully acknowledged. More detailed technical reports on this research are available to Industrial Members of Mobile VCE. We wish to thank Dr. Nick Papadoglou from Vodafone, United Kingdom, for his insightful comments.

References

[1] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Trans. Soft. Eng.*, vol. 24, no. 5, 1998, pp. 342–61.
[2] A. Bieszczad, B. Pagurek, and T. White, "Mobile Agents for Network Management," *IEEE Commun. Survey*, vol. 1, no. 1, 4th qtr. 1998.
[3] G. Goldszmidt and Y. Yemini, "Delegated Agents for Network Management," *IEEE Commun. Mag.*, vol. 36 no. 3, Mar. 1998.
[4] J. Philippe, M. Flatin, and S. Znaty, "Two Taxonomies of Distributed Network and System Management Paradigms," *Emerging Trends and Challenges in Network Management*, S. Erfani and P. Ray, Eds., Plenum, 2000.
[5] M. S. Daskin, *Network and Discrete Location*, Wiley, 1995.
[6] A. Liotta, "Towards Flexible and Scalable Distributed Monitoring with Mobile Agents," Ph.D. thesis, Dept. Comp. Sci., Univ. College London, U.K., July 2001.
[7] E. W. Zegura, K. L. Calvert, and M. J. Donahoo, "A Quantitative Comparison of Graph-Based Models for Internet Topology," *IEEE/ACM Trans. Net.*, 1997.
[8] K. L. Calvert, M. B. Doar, and E. W. Zegura, "Modeling Internet Topology," *IEEE Commun. Mag.*, June 1997.
[9] K. Fall and K. Varadhan, NS Notes and Documents, UC Berkeley, (http://www.isi.edu/ ~salehi/ ns_doc/), Oct. 1999.
[10] G. Knight and R. Hazemi, "Mobile Agent Based Management in the INSERT Project," *J. Net. Sys. Mgmt.*, vol. 7, no. 3, Sept. 1999.

Biographies

ANTONIO LIOTTA (a.liotta@eim.surrey.ac.uk) obtained his first degree in electronic engineering from the University of Pavia in 1994, an M.Sc. in information technology from Polytechnics of Milan in 1995, and a Ph.D. in computer science from University College London in 2001. Since 2000 he has worked at the University of Surrey as a researcher and academic, and has been involved in several collaborative projects in the area of network and system management. Recent research interests have included the use of mobile agents for distributed network monitoring, service management for the virtual home environment, and middleware for advanced services in 3G mobile systems.

GEORGE PAVLOU (G.Pavlou@eim.surrey.ac.uk) is professor of communication and information systems at the Centre for Communication Systems Research, School of Electronics and Computing, University of Surrey, United Kingdom, where he leads the activities of the Networks Research Group. He received a diploma in electrical engineering from the National Technical University of Athens, Greece, and M.Sc. and Ph.D. degrees in computer science from University College London. His research interests include network dimensioning, traffic engineering, ad hoc networks, code mobility, programmable and active networks, multimedia service control, and communications middleware. He has contributed to standardization activities in ISO, ITU-T, TMF, OMG, and IETF.

GRAHAM KNIGHT (g.knight@cs.ucl.ac.uk) obtained a B.Sc. in mathematics from Southampton University in 1969 and an M.Sc. in computer science from University College London in 1980. He has since worked at UCL as a researcher and academic. He has led UCL teams in several collaborative projects in the areas of network and system management, and ISDN. Recent research interests have included the use of mobile agents for service management, 3G mobile systems, and mobile multicast IP.